UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/670,802	09/26/2003	Victor Yodaiken	40101/13901 (2008.005)	2648

30636 7590 07/02/2007

FAY KAPLUN & MARCIN, LLP
150 BROADWAY, SUITE 702
NEW YORK, NY 10038

EXAMINER	
VO, TED T	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
07/02/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/670,802	YODAIKEN ET AL.	
	Examiner	Art Unit	
	Ted T. Vo	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *02 April 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-7 and 10-68* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-7 and 10-68* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is in response to the communication filed on 02/19/2004.

Claims 1-7, 10-68 are pending in the application.

Response to Arguments

2. In view of Applicants' argument to the objection to the specification related to the continuity data, this objection is withdrawn.

With regards to the arguments to the rejection to Claims 1-7, and 10-68, as anticipated by US Patent 5,136,709, the arguments have been fully considered, but Applicants failed to address the claims and the reference in the manner in the interview summary on 01/09/07, i.e., Applicants were noted in the Interview Summary that they would provide an amendment and address the patentable difererences between the claims and the US Pat. No. 5,136,709.

However, in the remarrks, Applicants fail to amend their claims and fail to address patentable features; Applicants, instead, argued that

> Shirakabe describes a method for generating an operating system. A driver 5, a linkage library 12, and a driver definition table 14 are combined by the use of a static linkage editor 7. (See Shirakabe, col. 5, !1.7-11 ; Fig. 1 .) The output of this combination is load module 26. (See id.) The Examiner contends that the process performed by the static linkage editor 7 and illustrated in Fig. 1 of Shirakabe anticipates the build system of claim 1. (See 11/3/06 Office Action, p, 4, 11.20-27.) However, the static linkage editor 7 is different from the build system of claim 1.
> First, once the build system of claim 1 has constructed the loadable module, it continues by "constructing a standard executable program from the loadable module and an execution library." Conversely, Shirakabe does not disclose performing further modifications to the load module 26; rather, it is used as a finished product. (See Shirakabe, cols. 5-7.) Further, Shirakabe does not state that the load module 26 is executable, nor does it discuss the creation of an
> executable program from the load module 26.
> Theretbre, it is respectfully submitted that Shirakabe does not disclose "a build system for constructing a loadable module from the application code and the environment library and for constructing a standard executable program from the loadable module and an execution library," as recited in claim 1. 'Accordingly, the rejection of claim 1 should be withdrawn. Because claims 2-4 and 10-20 depend from, and, therefore, include all of the

limitations of claim 1, it is respectfully submitted that these claims are also allowable for at least the reasons stated above.

Examiner responds: Applicants argued, Shiraable does not disclose "a build system for constructing a loadable module from the application code and the environment library and for constructing a standard executable program from the loadable module and an execution library," as recited in claim 1, where what the argument is only a generic allegation and unrelated.

For example, Applicant argued, "the static linkage editor 7 is different from the build system of claim 1"; and alleged the differences are:

- Applicant alleged, "Shirakabe does not disclose performing further modifications to the load module 26; rather, it is used as a finished product. (See Shirakabe, cols. 5-7.)";

This allegation is not related to the claims. It should be noted that with the editor, a user using the system of Figure 1 can write anything in the kernel. In fact, FIG 1 show an example of modification such as a code segment PRGNT Id(=m), has been modified and loaded in a location in a Driver (a) code in the kernel.

- Applicant alleged, "Shirakabe does not state that the load module 26 is executable, nor does it discuss the creation of an executable program from the load module 26".

It should be noted that kennel is a part of operation system resided in memory at all times to provide a basic service. The code in the kernel or what ever code that is called or linked is executable code. If Applicants say code loaded in the kernel such as #26 cannot executable, than Applicant are respectfully requested proving.

These Applicants' reasons are unable to show any patentable differences from their claims to the teaching of the reference.

Shirakabe disclose a system for dynamically linking application code (Object Modules #1, including Drivers) created by a programmer into a running operating system kernel (Kernel # 4), comprising:
an environment library comprising one or more routines for insulating the application code from the operating system environment and for implementing a uniform execution environment (i.e. that a storage area in the computer editable by the static linkage editor, for example a library expressed as "linkage Library 12", that stores drivers, subroutines. Each driver or subroutine is insulated from the object modules, but linkable with the kernel by subroutine calls in the object modules and editable by Static Linkage Editor #7); and

a build system for constructing a loadable module (that is Drier a, Driver b, that is loaded with loadable module # 26) ***from the application code and the environment library and for constructing a standard executable program from the loadable module and an execution library*** (Fig. 1 with Static Linkage Editor #7), ***wherein***

the execution library comprises one or more routines (such as drivers, linkage library shown in the left side of FIG. 1) ***for transparently loading the loadable module into the running operating system kernel*** (acting under the editor #7), ***passing arguments to the loadable module*** (e.g. Call KSUBm → PRGNT Id(=m) , ***and terminating and unloading the loadable module after receiving a termination signal*** (i.e. the termination perform by the user based on basis operations upon the Editor #7 or return command).

On the other hand, the limitation such as ***environment library*** and/or ***execution library*** does not present any novelty. As it is very well known in the art that any computer is operated with a file system that includes various types of libraries. A standard editor (simply like a text editor) is always provided with a connection to any library in the computer. Editor in the computer means that a user can modify whatever it is transparent under the editor. In the manner of the reference, Object modules in the left of FIG. 1 can modifiable and code in the kernel in the right side of FIG. 1 can writable. If Applicants argued that their differences are having "Libraries" and these are patent features and regard as invention then Examiner would like to respond that many prior arts don't want to mention them (***environment library***; i.e. a directory in which an application and editor reside, ***execution library***: i.e. a directory in which files like ".dll" reside) because they are so common.

It should be noted that Claims 5-7, 47-51 also further rejected under 35 U.S.C. 102(b) as being anticipated by Kempf et al, "Cross-Address Space Dynamic Linking", and Applicants are moot under this rejection.

Accordingly, the Applicants' reply is not fully responsive to the prior Office Action. The Acion is made FINAL.

Claim Rejections - 35 USC § 102

3. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

4. Claims 1-7, 10-68 are rejected under 35 U.S.C. 102(b) as being anticipated by Shirakable et al., US Pat No. 5,136,709 A.

Given the broadest reasonable interpretation of followed claims in light of the specification.

As per claim 5: Shirakabe discloses,

A method, comprising: creating a loadable module; See FIG. 1

creating an executable program;

and executing the executable program, wherein the executable program performs a method comprising the steps of:

setting up input/output channels;

inserting the loadable module into an operating system address space, wherein, once the loadable the module is inserted into the operating system address space, the loadable module begins to execute; and waiting for the loadable module to connect via kernel/user channels and then connecting those kernel/user channels to the input/output channels.

(Shirakabe mentions prior art acts: first of all a user creates a loadable program in his computer environment, where the loadable program (such as an object module or load module in FIG. 1) needs loaded in an execution/or running kernel. The user simply generates an executable program that issues input/output request such as in FIG. 4, or 8 (within the claims is *setting up input/output channels*). The execution of the program will map the loadable program, according to input/output request, into the address space (see col. 1). In the disclosure, Shirakabe discloses an executable program (algorithm in FIGs 4-5, the program embedded in an editor of FIG. 1. An Since the loading is executed by the executable program, it needs time so that the execution encounter the instructions that perform the loading – in the reference it shows each driver is included with open and close routines (FIG. 8)).

As per claim 6: Shirakabe discloses, *The method of claim 5, wherein after the loadable module is inserted into the operating system address space the loadable module performs a method comprising the steps of: creating kernel/user channels* (See FIG. 1, FIG. 8);

creating a thread to execute application code (e.g. see col. 6:27:30, or col. 5:45-46: CALL KSUBm);

and waiting for the thread to complete. E.g. see FIG. 1, an address space YYY is call and RETURN);

As per claim 7: Shirakabe discloses, *The method of claim 6, wherein the method performed by the loadable module further includes the step of freeing resources after the thread completes.*

It recites a principle of execution. For example, a user finishes his job – See FIG. 6, OPEN/CLOSE routines to perform the opening or completion.

As per Claim 1: Shirakabe discloses,

A system for dynamically linking application code created by a programmer into a running operating system kernel, comprising:

an environment library comprising one or more routines for insulating the application code from the operating system environment and for implementing a uniform execution environment; (see FIG. 1, the computer environment implement the linkage editor)

and a build system for constructing a loadable module from the application code and the environment library and for constructing a standard executable program from the loadable module and an execution library, wherein the execution library comprises one or more routines for transparently loading the loadable module into the running operating system kernel, passing arguments to the loadable module, and terminating and unloading the loadable module after receiving a termination signal (See FIG. 1. it is a built system for constructing Load Modules (2), including Linkage Library, where a loadable program is transparently loading via a Linkage editor. See the process in FIG. 1 from Linage library, 12, to 7, to 6, to 16).

As per Claim 2: Shirakabe discloses, *The system of claim 1, further comprising an infrastructure library comprising one or more routines executed prior to loading the loadable module into the running operating system kernel and/or after unloading the loadable module from the kernel.* (It should be noted that in a standard computer, a directory of stored files is a library: using the languages such as *environment*

library, an *execution library*, infrastructure library will not be distinguishable from the memory storage or shared libraries that existed in every computer system. The reference is in a computer environment that includes such memory storage, shared libraries, where every program in this storage can be executed for verification before loaded).

As per Claim 3: Shirakabe discloses, *The system of claim 1, wherein the execution library includes one or more routines for setting up input/output channels* (See FIG. 4 or 8).

As per Claim 4: Shirakabe discloses, *The system of claim 1, wherein the standard executable program may be in several files or a single file* (It just a program).

As per claim 10: Shirakabe discloses, *The system of claim 1, wherein one of the one or more routines of the execution library includes code for executing a utility for installing the loadable module into the running operating system kernel* (See FIG. 1, where "editor" has means for *executing a utility for installing the loadable module*).

As per Claim 11: Shirakabe discloses, *The system of claim 10, wherein the utility for installing the loadable module into the running operating system kernel is the insmod program* (FIG 1, with Linkage editor has either means of "insmod" or "mesg", or "rmmod" as given broad interpretation in light of the specification).

As per Claim 12: Shirakabe discloses, *The system of claim 1, wherein the build system includes instructions for compiling the application code into object code* (The loadable module is executable code (FIG 1, "Object Modules"); in a computer, ever Object code/executable code must be from compilation).

As per Claim 13: Shirakabe discloses, *The system of claim 12, wherein the build system further includes instructions for linking said object code with object code from the environment library to produce a linked object module.* (See in FIG. 1)

As per Claim 14: Shirakabe discloses, *The system of claim 13, wherein the build system further includes instructions for converting the linked object module into a C code array* (OBJECT MODULES has means of C code Array).

As per Claim 15: Shirakabe discloses, *The system of claim 13, wherein the build system further includes instructions for compiling the C code array to produce an object file and for linking said object file with*

object code from the execution library to produce the standard executable program (OBJECT MODULES in FIG 1, has means of being compiled from instructions in a compiler).

As per Claim 16: Shirakabe discloses, *The system of claim 1, wherein the environment library includes one or more routines to create kernel/user channels* (Claiming routines is claiming program per se, where the reference shows such limitations in FIG. 8).

As per Claim 17: Shirakabe discloses, *The system of claim 1, wherein the environment library includes one or more routines to create a thread to execute the application code.* (Claiming routines is claiming program per se, where the reference shows OBJECT MODULES are of an application and these MODULES are executable).

As per Claim 18: Shirakabe discloses, *The system of claim 17, wherein the environment library includes one or more routines for freeing resources and unloading the loadable module when the thread completes.*

(See FIG 8, for example, OPEN or CLOSE routines – Note: the limitation can be seen from basis commands in a standard editor such as "save", "close", or "exit", etc.)

As per Claim 19: Shirakabe discloses, *The system of claim 1, wherein the environment library includes one or more routines for (a) copying in arguments; (b) creating communication channels that connect the loadable module to the executable program; (c) requesting a block of memory from the operating system and storing a structure therein that describes the application code; and (d) putting data describing the application code on a task list.* (See FIGs 1-8, Claimed functionality is a performance through repeated limitations of Claim 1).

As per Claim 20: Shirakabe discloses, *The system of claim 19, wherein the environment library further includes one or more routines for (a) removing said data describing the application code from the task list; (b) closing said communication channels that connect the loadable module to the executable program; and (c) freeing the block of memory that was requested from the operating system.*

(See FIG 8, for example, OPEN or CLOSE routines – Note: the limitation can be seen from basis commands in a standard editor such as "save", "close", or "exit", etc.)

As per Claim 21: Shirakabe discloses, *A computer readable medium having computer instructions stored thereon, the computer instructions comprising: a first set of computer instructions for insulating application code from an operating system environment; a second set of computer instructions for constructing a loadable module from the application code and the first set of computer instructions; and a third set of computer instructions for constructing an executable program from the loadable module and a fourth set of computer instructions; wherein the fourth set of computer instructions includes computer instructions for transparently loading the loadable module into a running operating system kernel, passing arguments to the loadable module, and terminating and unloading the loadable module from the running operating system kernel after receiving a termination signal.*

The claim is a computer readable medium in which the functionality performs the method in Claim 1. See rationale addressed in Claim 1 above for the rejection of Claim 21.

As per Claim 22: Shirakabe discloses, *The computer readable medium of claim 21, wherein the computer instructions for loading the loadable module into the running operating system kernel include computer instructions for executing a utility for installing the loadable module into the running operating system kernel.* See rationale addressed in Claim 10 above for the rejection of Claim 22.

As per Claim 23: Shirakabe discloses, *The computer readable medium of claim 22, wherein the utility for installing the loadable module into the running operating system kernel is the insmod program.*

See rationale addressed in Claim 11 above for the rejection of Claim 23. Note: "insmod program" is subjected to discussion in MPEP 2115. A limitation used in the claim must impart functionality, otherwise it is treated as material worked upon and does not limit the claim.

As per Claim 24: Shirakabe discloses, *The computer readable medium of claim 21, wherein the second set of computer instructions includes instructions for compiling the application code into object code.* See rationale addressed in Claim 12 above for the rejection of Claim 24.

As per Claim 25: Shirakabe discloses, *The computer readable medium of claim 24, wherein the second set of computer instructions further includes instructions for linking said object code with object code from the environment library to produce a linked object module.* See rationale addressed in Claim 13 above for the rejection of Claim 25.

As per Claim 26: Shirakabe discloses, *The computer readable medium of claim 25, wherein the third set of computer instructions includes instructions for converting the linked object module into a C code array.* See rationale addressed in Claim 14 above for the rejection of Claim 26.

As per Claim 27: Shirakabe discloses, *The computer readable medium of claim 26, wherein the third set computer instructions of further includes instructions for compiling the C code array to produce an object file and for linking said object file with object code from a library to produce the executable program.* See rationale addressed in Claim 15 above for the rejection of Claim 27.

As per Claim 28: Shirakabe discloses, *The computer readable medium of claim 21, wherein the first set of computer instructions includes instructions for creating kernel/user channels.* See rationale addressed in Claim 16 above for the rejection of Claim 28.

As per Claim 29: Shirakabe discloses, *The computer readable medium of claim 21, wherein the first set of computer instructions includes instructions for creating a thread to execute the application code.* See rationale addressed in Claim 17 above for the rejection of Claim 29.

As per Claim 30: Shirakabe discloses, *The computer readable medium of claim 29, wherein the first set of computer instructions includes instructions for freeing resources and unloading the loadable module when the thread completes.* See rationale addressed in Claim 18 above for the rejection of Claim 30.

As per Claim 31: Shirakabe discloses, *The computer readable medium of claim 21, wherein the first set of computer instructions includes instructions for (a) creating communication channels that connect the loadable module to the executable program; (b) requesting a block of memory from the operating system and storing a structure therein that describes the application code; and (c) putting data describing the application code on a task list.* See rationale addressed in Claim 19 above for the rejection of Claim 31.

As per Claim 32: Shirakabe discloses, *The computer readable medium of claim 31, wherein the first set of computer instructions further includes instructions for (a) removing said data describing the application code from the task list; (b) closing said communication channels that connect the loadable module to the executable program; and (c) freeing the block of memory that was requested from the operating system.* See rationale addressed in Claim 20 above for the rejection of Claim 32.

As per Claim 33: Shirakabe discloses, *The computer readable medium of claim 21, wherein the executable program may be in several files or a single file.*

See rationale addressed in Claim 4 above for the rejection of Claim 33.

As per Claim 34: Shirakabe discloses, *A computer system, comprising: first means for insulating application code from an operating system environment; second means for constructing a loadable module from the application code and the first means; third means for constructing an executable program from the loadable module; and fourth means for transparently loading the loadable module into a running operating system kernel, passing arguments to the loadable module, and terminating and unloading the loadable module from the running operating system kernel after receiving a termination signal.*

The claim is a system in which the functionality performs the method in Claim 1. See rationale addressed in Claim 1 above for the rejection of Claim 34.

As per Claim 35: Shirakabe discloses, *The computer system of claim 34, wherein means for loading the loadable module into the running operating system kernel include means for executing a utility for installing the loadable module into the running operating system kernel.* See rationale addressed in Claim 10 above for the rejection of Claim 35.

As per Claim 36: Shirakabe discloses, *The computer system of claim 35, wherein the utility for installing the loadable module into the running operating system kernel is the insmod program.*

See rationale addressed in Claim 11 above for the rejection of Claim 36. Note: "insmod program" is subjected to discussion in MPEP 2115. A limitation used in the claim must impart functionality, otherwise it is treated as material worked upon and does not limit the claim.

As per Claim 37: Shirakabe discloses, *The computer system of claim 34, wherein the second means includes means for compiling the application code into object code.* See rationale addressed in Claim 12 above for the rejection of Claim 37.

As per Claim 38: Shirakabe discloses, *The computer system of claim 37, wherein the second means further includes means for linking said object code with object code from the environment library to produce a linked object module.* See rationale addressed in Claim 13 above for the rejection of Claim 38.

As per Claim 39: Shirakabe discloses, *The computer system of claim 38, wherein the third means includes means for converting the linked object module into a C code array.* See rationale addressed in Claim 14 above for the rejection of Claim 39.

As per Claim 40: Shirakabe discloses, *The computer system of claim 39, wherein the third means further includes instructions for compiling the C code array to produce an object file and for linking said object file with object code from a library to produce the executable program.* See rationale addressed in Claim 15 above for the rejection of Claim 40.

As per Claim 41: Shirakabe discloses, *The computer system of claim 34, wherein the first means includes means for creating kernel/user channels.* See rationale addressed in Claim 16 above for the rejection of Claim 41.

As per Claim 42: Shirakabe discloses, *The computer system of claim 34, wherein the first means includes means for creating a thread to execute the application code.* See rationale addressed in Claim 17 above for the rejection of Claim 42.

As per Claim 43: Shirakabe discloses, *The computer system of claim 42, wherein the first means includes means for freeing resources and unloading the loadable module when the thread completes.* See rationale addressed in Claim 18 above for the rejection of Claim 43.

As per Claim 44: Shirakabe discloses, *The computer system of claim 34, wherein the first means includes means for (a) creating communication channels that connect the loadable module to the executable program; (b) requesting a block of memory from the operating system and storing a structure therein that describes the application code; and (c) putting data describing the application code on a task list.*
See rationale addressed in Claim 19 above for the rejection of Claim 44.

As per Claim 45: Shirakabe discloses, *The computer system of claim 44, wherein the first means further includes means for (a) removing said data describing the application code from the task list; (b) closing said communication channels that connect the loadable module to the executable program; and (c) freeing the block of memory that was requested from the operating system.*
See rationale addressed in Claim 20 above for the rejection of Claim 45.

As per Claim 46: Shirakabe discloses, *The computer system of claim 34, wherein the executable program may be in several files or a single file.* See rationale addressed in Claim 4 above for the rejection of Claim 46.

As per Claim 47: Shirakabe discloses, *A computer system for dynamically linking application code created by a programmer into a running operating system kernel, comprising: means for creating a loadable module; and means for creating an executable program that is configured to performs a method comprising the steps of: setting up input/output channels; inserting the loadable module into address space of the running operating system kernel, wherein, once the loadable the module is inserted into the address space, the loadable module begins to execute; and waiting for the loadable module to connect via kernel/user channels and then connecting those kernel/user channels to the input/output channels.* See rationale addressed in Claim 5 above for the rejection of Claim 47.

As per Claim 48: Shirakabe discloses, *The computer system of claim 47, wherein the loadable module is configured to perform a method after the loadable module is inserted into the operating system address space, wherein said method comprises the steps of: creating kernel/user channels; creating a thread to execute the application code; and waiting for the thread to complete.* See rationale addressed in Claim 6 above for the rejection of Claim 48.

As per Claim 49: Shirakabe discloses, *The computer system of claim 48, wherein the method performed by the loadable module further includes the step of freeing resources after the thread completes.* See rationale addressed in Claim 7 above for the rejection of Claim 49.

As per Claim 50: Shirakabe discloses, *The computer system of claim 47, wherein the step of inserting the loadable module into an operating system address space includes the step of creating a child process, wherein the child process replaces its image with the insmod process image.* (See the editor using in FIGs 1, 8).

As per Claim 51: Shirakabe discloses, *The computer system of claim 50, wherein the step of inserting the loadable module into an operating system address space further includes the step of piping the loadable module to the insmod process.* (See the editor using in FIGs 1, 8).

As per Claim 52: Shirakabe discloses, *A method for dynamically linking application code created by a user into a running operating system kernel, comprising:*

constructing a loadable module from application source code written by a user (See FIG. 1, Shirakabe shows loadable modules are created within libraries seen via an Editor);

creating an executable program, wherein the executable program is configured to transparently load the loadable module into the running operating system kernel (See FIG. 8, Shirakabe shows transparently loading the loadable modules as noted in FIG. 1, into the running operating system;

executing the executable program, thereby loading the loadable module into the running operating system kernel (See Col. 5:24-65);

and unloading the loadable module from the running operating system kernel by sending a termination signal to the executable program (See FIG. 8, provided with OPEN/CLOSE routines).

As per Claim 53: Shirakabe discloses, *The method of claim 52, wherein the application source code is an ordinary application program.* (The OBJECT MODULES shown in FIG 1 are loadable modules of an application that could be original).

As per Claim 54: Shirakabe discloses, *The method of claim 52, wherein the step of constructing the loadable module from the application source code consists essentially of executing a pre-defined makefile.*

(Such OBJECT MODULES shown in FIG 1 are pre-executable before loaded).

As per Claim 55: Shirakabe discloses, *The method of claim 52, further comprising the step of providing a makefile to the user, wherein the user performs the step of constructing the loadable module by executing the makefile after the user has created the application code.*

This claim is associated with user intervention; every acts performed by a user is prior art. The reference is operable by a user via the editor in term of choice. For example, a user uses an editor to create a file, to compile the file, to execute the file. This is a basis of program generation.

As per Claim 56: Shirakabe discloses, *The method of claim 52, further comprising the step of providing the user with a library comprising object code, wherein the step of constructing the loadable module from the application source code comprises the steps of compiling the application source code into object*

code; linking the object code with object code from the library to produce a linked object module; and converting the linked object module into a C code array. (See FIG. 1: OBJECT MODULES, and rationale addressed in Claim 14).

As per Claim 57: Shirakabe discloses, *The method of claim 56, wherein the step of constructing the loadable module further comprises the step of compiling the C code array to produce an object file.* See rationale addressed in Claim 15.

As per Claim 58: Shirakabe discloses, *The method of claim 57, further comprising the step of providing the user with a second library comprising object code, wherein the step of constructing the executable program comprises the steps of linking the object file with object code from the second library* (See FIG. 1 or FIG. 8).

As per Claim 59: Shirakabe discloses, *The method of claim 56, wherein the library includes one or more routines to create kernel/user channels.* (See FIG. 1, FIG. 8);

As per Claim 60: Shirakabe discloses, *The method of claim 56, wherein the library includes one or more routines to create a thread to execute the application code* (e.g. see col. 6:27:30, or col. 5:45-46: CALL KSUBm);

As per Claim 61: Shirakabe discloses, *The method of claim 60, wherein the library includes one or more routines for freeing resources and unloading the loadable module when the thread completes* (E.g. see FIG. 1, an address space YYY is call and RETURN).

As per Claim 62: Shirakabe discloses, *The method of claim 56, wherein the library includes one or more routines for (a) copying in arguments; (b) creating communication channels that connect the loadable module to the executable program; (c) requesting a block of memory from the operating system and storing a structure therein that describes the application code; and (d) putting data describing the application code on a task list.* See rationale addressed in Claim 19 above.

As per Claim 63: Shirakabe discloses, *The method of claim 62, wherein the environment library further includes one or more routines for (a) removing said data describing the application code from the task list; (b) closing said communication channels that connect the loadable module to the executable program;*

and (c) freeing the block of memory that was requested from the operating system. See rationale addressed in Claim 20 above.

As per Claim 64: Shirakabe discloses, *The method of claim 52, wherein the executable program is configured to set up input/output channels.* See rationale addressed in Claim 3 above.

As per Claim 65: Shirakabe discloses, *The method of claim 52, wherein the executable program is configured to execute a utility for installing the loadable module into the running operating system kernel.* See rationale addressed in Claim 10 above.

As per Claim 66: Shirakabe discloses, *The method of claim 65, wherein the utility for installing the loadable module into the running operating system kernel is the insmod program.* See rationale addressed in Claim 11 above.

As per Claim 67: Shirakabe discloses, *The method of claim 65, wherein the step of inserting the loadable module into an operating system address space includes the step of creating a child process, wherein the child process replaces its image with the insmod process image.* (See FIG. 1, FIG. 8).

As per Claim 68: Shirakabe discloses, *The method of claim 67, wherein the step of inserting the loadable module into an operating system address space further includes the step of piping the loadable module to the insmod process* (See FIG. 1, FIG. 8).

5. Claims 5-7, 47-51 are rejected under 35 U.S.C. 102(b) as being anticipated by Kempf et al, "Cross-Address Space Dynamic Linking".

Given the broadest reasonable interpretation of followed claims in light of the specification.

As per claim 5: Kempf discloses,

A method, comprising: creating a loadable module; creating an executable program;

and executing the executable program, (See Abstract, p. 0)

wherein the executable program performs a method comprising the steps of:

setting up input/output channels (See Figure 2: Code Table/Target Domain Objects);

inserting the loadable module into an operating system address space, wherein, once the loadable the module is inserted into the operating system address space, the loadable module begins to execute; and

waiting for the loadable module to connect via kernel/user channels and then connecting those kernel/user channels to the input/output channels (See Abstract, and sec. 5, started at p. 6 and the rest of the reference

As per claim 6: Shirakabe discloses, *The method of claim 5, wherein after the loadable module is inserted into the operating system address space the loadable module performs a method comprising the steps of: creating kernel/user channels* (See p.2, Previous Work, enter kernel mode, etc.);
creating a thread to execute application code (See p.2, Previous Work, e.g., thread running, etc.);
and waiting for the thread to complete. (An execution is always included with start/finish, simply performed via call/return, and terminological reference as "invocation". See reference, p. 6: indentation 6. and section 5, e.g., process is finish, etc.).

As per claim 7: Shirakabe discloses, *The method of claim 6, wherein the method performed by the loadable module further includes the step of freeing resources after the thread completes.*
(See reference, p. 6: indentation 6. and section 5, e.g., process is finish, etc.).

As per Claims 47-49: See Rationale addressed in Claim 5-7.

As per Claim 50: Kempf discloses, *The computer system of claim 47, wherein the step of inserting the loadable module into an operating system address space includes the step of creating a child process, wherein the child process replaces its image with the insmod process image.* (See p. 6, indentation 5).

As per Claim 51: Shirakabe discloses, *The computer system of claim 50, wherein the step of inserting the loadable module into an operating system address space further includes the step of piping the loadable module to the insmod process.* (See p. 6, indentation 5, and associated with the Figure 1 and Figure 2).

Conclusion

6. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date

of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ted T. Vo whose telephone number is (571) 272-3706. The examiner can normally be reached on 8:00AM to 4:30PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Y. Zhen can be reached on (571) 272-3708.

The facsimile number for the organization where this application or proceeding is assigned is the Central Facsimile number **571-273-8300**.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100. Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

TTV
June 22, 2007

TED VO
PRIMARY EXAMINER